BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

⑲日本国特許庁（JP）　⑪特許出願公開

⑫公開特許公報（A）　昭62－283496

⑤1Int.Cl.⁴	識別記号	庁内整理番号	⑬公開　昭和62年(1987)12月9日
G 11 C 17/00	307	6549－5B	

審査請求　未請求　発明の数 1　（全9頁）

⑤4発明の名称　プログラマブルリードオンリメモリの書き込み回数管理方式

⑳特　願　昭61－124731
㉒出　願　昭61(1986)5月31日

㉗発明者　仲田　眞一　東京都大田区下丸子3丁目30番2号　キヤノン株式会社内
㉑出願人　キヤノン株式会社　東京都大田区下丸子3丁目30番2号
㉔代理人　弁理士　小林　将高

明　細　書

1．発明の名称

プログラマブルリードオンリメモリの書き込み回数管理方式

2．特許請求の範囲

記憶領域に書き込まれた情報を電気的に消去可能なプログラマブルリードオンリメモリにおいて、前記記憶領域を複数のブロックに分割し、各ブロック毎に書き込み回数を記憶し、この書き込み回数に応じて書き込み頻度の低いブロックを未使用ブロックの先頭から書き込むとともに、前記書き込み頻度の低いブロックを未使用ブロックの最後尾に接続させることを特徴とするプログラマブルリードオンリメモリの書き込み回数管理方式。

3．発明の詳細な説明

〔産業上の利用分野〕

この発明は、電気的消去可能なプログラマブルリードオンリメモリの書き込み回数の管理方式に関するものである。

〔従来の技術〕

従来のＥＥＰＲＯＭ（Electrical Eresable and Programmable ROM）は、容量も少なく、また書き込むために必要な外部回路が多かった。さらに、チップ内のすべてのデータを消去するモードしか有していなかった。最近は、容量も大きくなるとともに、外部回路も殆ど必要なくＣＰＵのアドレスバス，データバスに結線できるようになり、またＥＥＰＲＯＭ内の1バイトのデータのみの消去も可能となってきた。以上の改良により、使用目的によっては、従来のランダムアクセスメモリ（ＲＡＭ）で構成していた機能の置換が可能となった。

例えば、従来の小型パソコン、日本語ワープロで作成したプログラムや文章，外字等を保存しておくためにメモリカードと云うものがある。これは、必要なときにパソコン，日本語ワープロ等の本体に差し込んでプログラムや文章を記憶させ、本体から引き抜いても、そのデータを記憶しているように、メモリカード内にはＲＡＭと電池が搭

載されていた。そこで、メモリカードをＥＥＰＲＯＭで構成することにより、電池を無くすることができると考えられた。

〔発明が解決しようとする問題点〕

ところが、ＥＥＰＲＯＭでは従来のＲＡＭのように自由に何度も書き換えられない制約があり、すなわち、あらかじめ設定される書き込み回数を越えて、メモリカードへの書き込みを行うことにより、記憶しているはずのデータを消失させてしまう等の問題点があった。またＥＥＰＲＯＭに書き込まれたデータのうち、頻繁に書き換えられるデータと書き換え頻度の少ないデータとが存在し、書き換え頻度の高いデータの書き換え回数が所定値を越えると、ＥＥＰＲＯＭへの書き換えが可能にも関わらず書き換え不能となる問題点があった。

この発明は、上記の問題点を解消するためになされたもので、ＥＥＰＲＯＭに書き込まれるデータの消失を防止するとともに、ＥＥＰＲＯＭへの書き込み回数を平均化させるとともに、ＥＥＰＲＯＭ上の書き換え頻度を平均化して、ＥＥＰＲＯＭへの書き換え寿命を延命できるプログラマブルリードオンリメモリの書き込み回数管理方式を得ることを目的とする。

〔問題点を解決するための手段〕

この発明に係るプログラマブルリードオンリメモリの書き込み回数管理方式は、記憶領域を複数のブロックに分割し、各ブロック毎に書き込み回数を記憶し、この書き込み回数に応じて書き込み頻度の低いブロックを未使用ブロックの先頭から書き込むとともに、前記書き込み頻度の低いブロックを未使用ブロックの最後尾に接続させる。

〔作用〕

この発明においては、記憶領域の各ブロック毎の書き込み回数を記憶しておき、この書き込み回数に応じて書き込み頻度の低いブロックを未使用ブロックの先頭から書き込ませるとともに、書き込み頻度の低いブロックを未使用のブロックの最後端に接続させる。

〔実施例〕

第１図（ａ）はこの発明の一実施例を示すプログラマブルリードオンリメモリへの書き込み回数管理方式を説明する模式図であり、１はＥＥＰＲＯＭで、例えば書き込み容量が32798 バイト×８ビットで、書き込み回数が１万回に設定してある。ＥＥＰＲＯＭ１は、ポインタブロック１ａおよび予備ポインタブロックＳＰＢ１～ＳＰＢ５０より構成される。ポインタブロック１ａは４アドレス（各１バイト）で構成され、『０～１』番地の２バイトで、書き換え回数ＷＣＮＴ、例えば『1388_{16}』を記憶している。またポインタブロック１ａの『２』番地の１バイトは、ディレクトリＤＢ、例えば『01_{16}』を記憶している。さらに、ポインタブロック１ａの『３』番地の１バイトは、未使用のスタートブロック番号ＯＳＢ、例えば『33_{16}』を記憶している。またポインタブロック１ａの『４』番地の１バイトは、未使用のエンドブロック番号ＯＥＢ、例えば『$8A_{16}$』を記憶している。

第１図（ｂ）はこの発明の装置構成の一例を説明するブロック図であり、１１はＣＰＵで、ＲＯＭ１１ａ，ＲＡＭ１１ｂを有し、ＲＯＭ１１ａに格納された第７図，第８図に示すフローに準じたプログラムに応じて各部を制御する。１２は入力手段で、データ書き込み装置１３にセットされるＥＥＰＲＯＭ１へのデータ書き込みおよびデータ消去を指示する。なお、ＣＰＵ１１にはデータの転送を行うアキュムレータＡＣＣ，ＢＣＣを有している。

第２図は第１図（ａ）に示すＥＥＰＲＯＭ１の構造を示す模式図であり、２１はブロック番号であり、例えば１２７個のブロックＢＬＯＣＫ１～ＢＬＯＣＫ１２７に分割されている。各ブロックは、例えば２５６バイトで構成され、先頭の２バイトで、そのブロックが更新された回数、すなわち、後述する更新回数が記憶されている。次に続く２５３バイトは記憶データＤＡＴＡが記憶されており、最後の１バイトは、記憶データＤＡＴＡがこのブロックに留まるか、または他のブロックに及ぶかどうかを示す継続ブロックエリアＣＢが

あり、他のブロックに記憶データＤＡＴＡが及ぶ場合は、継続ブロックエリアＣＢには継続するブロック番号が記憶され、他のブロックに記憶データＤＡＴＡが及ばない場合は、継続ブロックエリアＣＢには『ＦＦ$_{16}$』が記憶されている。

第３図は第２図に示す各ディレクトリブロック構造を説明する模式図であり、３０は前記ディレクトリＤＢに指示されるディレクトリブロック、３１は前記ディレクトリブロック３０の更新カウンタで、例えば２バイトで構成される。３２はファイル領域で、各ファイル名が１２バイトで記憶される。３３はスタートブロック番号エリア（ＳＢ）で、例えば１バイトで構成され、ファイルのスタートブロック番号が記憶されている。３４はエンドブロック番号エリア（ＥＢ）で、例えば１バイトで構成され、ファイルのエンドブロック番号が記憶されている。３５はチェーンブロックエリア（ＣＢ）で、ディレクトリブロック３０に継続するディレクトリブロックの有無を記憶する。例えばチェーンブロックエリア３５が『ＦＦ$_{16}$』となる。なお、ディレクトリブロック３０は、例えば１８個のファイル領域３２で構成される。

次に第１図（ａ）および第３図を参照しながらＥＥＰＲＯＭ１の構造について説明する。

第１図（ａ）に示すようにポインタブロック１ａの書き換え回数ＷＣＮＴに、例えば『１３８８$_{16}$』が記憶されているとすると、５０００回の更新が行われたことを示し、またディレクトリＤＢには『０１$_{16}$』が記憶されているので、ディレクトリＤＢに指示されるディレクトリブロック３０のブロック番号が『１』で、そのディレクトリブロック３０の更新カウンタ３１には、『１４２Ｆ$_{16}$』が記憶されている。これは、このディレクトリブロック３０を５１６７回更新したことを示し、ファイル領域３２のファイル（ＦｉＬｅ）１（ファイル名）はスタートブロック番号エリア３３が『０２$_{16}$』で、エンドブロック番号エリア３４が『０５$_{16}$』となっているため、ブロックＢＬＯＣＫ２から始まり、ブロックＢＬＯＣＫ５で終ることになる。またファイル領域３２のファイル２は、スタートブロック番号エリア３３が『０Ａ$_{16}$』で、エンドブロック番号エリア３４が『０Ｆ$_{16}$』となっているため、ブロックＢＬＯＣＫ１０から始まり、ブロックＢＬＯＣＫ１５で終ることになる。さらに、ファイル領域３２のファイル３（ファイル名）は、スタートブロック番号エリア３３が『１５$_{16}$』で、エンドブロック番号エリア３４が『１８$_{16}$』となっているため、ブロックＢＬＯＣＫ２１から始まり、ブロックＢＬＯＣＫ２４で終ることになる。またファイル領域３３のファイル３の次に『ＦＦ$_{16}$』が書かれているので、このファイル領域３３はファイル３で終了していることになる。

第４図は未使用のＥＥＰＲＯＭ１の状態を説明する模式図であり、第１図（ａ），第３図と同一のものには同じ符号を付している。

この図から分かるように、未使用のＥＥＰＲＯＭ１のポインタブロック１ａの書き換え回数ＷＣＮＴが『０００１$_{16}$』、ディレクトリＤＢが『０１$_{16}$』、未使用のスタートブロック番号ＯＳＢが『０２$_{16}$』、未使用のエンドブロック番号ＯＥＢが『７Ａ$_{16}$』がそれぞれポインタブロック１ａの０番地から４番地にそれぞれ記憶されている。これにより、ディレクトリＤＢに指示されるブロックＢＬＯＣＫ１を参照すると、更新カウンタ３１に『０００１$_{16}$』が書き込まれているとともに、ファイル領域３２のファイル１に『ＦＦ$_{16}$』が書き込まれており、さらに、チェーンブロックエリア３５に『ＦＦ$_{16}$』が書き込まれており、ＥＥＰＲＯＭ１が未使用状態であることを示している。

さらに、ポインタブロック１ａのスタートブロック番号ＯＳＢおよびエンドブロック番号ＯＥＢには『０２$_{16}$』，『７Ｆ$_{16}$』がそれぞれ書き込まれている。すなわち、ブロックＢＬＯＣＫ２～１２７には先頭の２バイトに『０００１$_{16}$』が書き込まれ、最終の１バイトに各後続のブロックの継続を示すチェーンブロックエリア３５には、ブロックＢＬＯＣＫ２～１２６に対して『０３～７Ｆ$_{16}$』が書き込まれ、ブロックＢＬＯＣＫ１２７のチェーンブロックエリア３５には『ＦＦ』が書

き込まれている。このように、各ブロックＢＬＯＣＫ２～１２７は１つのチェイン構造となる。

次に第３図，第５図（ａ），（ｂ）を参照しながらＥＥＰＲＯＭ１への書き込み動作を説明する。

第５図（ａ），（ｂ）はＥＥＰＲＯＭ１への書き込み動作を説明する模式図であり、第１図（ａ），第３図と同一のものには同じ符号を付している。なお、書き込み直前は、第３図に示す状態であったものとする。

まず、各ブロックＢＬＯＣＫのファイル領域３２の先頭が『００$_{16}$』のところを探し当てる。第３図の場合は、ファイル２とファイル３との間に『００$_{16}$』があり、そこにファイル４という名前を１２バイトで書き込み、ポインタブロック１ａの未使用ブロックのスタートブロック番号ＯＳＢを参照して、スタートブロック番号ＯＳＢの指示するブロックＢＬＯＣＫ、すなわち『５７$_{16}$』の先頭の２バイト情報、すなわち、更新カウンタ３１を『１』インクリメントし、その加算値が、例えば１万回を越えているようであれば、ファイル４のチェーンブロックエリア３５が示すブロックＢＬＯＣＫに対して同様の操作を行い、更新カウンタ３１が１万回以下のブロックＢＬＯＣＫを探し当てて、そのブロックＢＬＯＣＫの番号をポインタブロック１ａのスタートブロック番号ＯＳＢに書き込むとともに、ファイル４のデータをブロックＢＬＯＣＫ８７（２５３バイト）に書き込み、ブロックＢＬＯＣＫ８７に溢れるようであれば、ブロックＢＬＯＣＫ８７のチェーンブロックエリア３５の指示するブロックＢＬＯＣＫの更新カウンタ３１を『１』インクリメントして加算値が、例えば１万回を越えているかどうかを調べ、指示されるブロックＢＬＯＣＫの更新カウンタ３１が１万回を越えるようであれば、更新回数が１万回以下のブロックＢＬＯＣＫを探し当て、そのブロックＢＬＯＣＫの番号を直前に書き込んだブロックＢＬＯＣＫのチェーンブロックエリア３５に書き込む。このようにして、データの書き込みが行われ、更新回数が１万回を越えるブロックＢＬＯＣＫが排除されて行く。そして、書き込みデータがなくなるまで同様の操作を行い、最後に書き込んだブロックＢＬＯＣＫのチェーンブロックエリア３５に記憶されていた内容を新しい未使用のスタートブロック番号ＯＳＢに書き換え、ポインタブロック１ａの書き換え回数ＷＣＮＴを『１』インクリメントして『１３８９$_{16}$』となり、最後にデータを書き込んだブロックＢＬＯＣＫのチェーンブロックエリア３５を『ＦＦ$_{16}$』にする。そして、ディレクトリブロック３０の最終ブロック番号を記憶するエンドブロック番号エリア３４に最後のデータを書き込んだブロックＢＬＯＣＫの番号を書き込むとともに、更新カウンタ３１を『１』インクリメントすると、第５図（ｂ）に示されるように、更新カウンタ３１が『１４３０$_{16}$』となり、ファイル４のスタートブロック番号エリア３３が『３３$_{16}$』で、エンドブロック番号エリア３４が『３７$_{16}$』となる。

次に第５図（ａ），（ｂ）を参照しながらＥＥＰＲＯＭ１に書き込まれているファイル１の削除動作について説明する。

ディレクトリブロック３０となるブロックＢＬＯＣＫ１よりファイル１を探し、ファイル領域３２の先頭の２バイトを『００$_{16}$』とする。次いで、ディレクトリブロック３０の更新カウンタ３１を『１』インクリメントし、ファイル１のスタートブロック番号エリア３３とエンドブロック番号エリア３４のデータを参照して、ポインタブロック１ａのエンドブロック番号ＯＥＢが指示するブロックのチェーンブロックエリア３５の内容（削除直前までは『ＦＦ$_{16}$』であった）をスタートブロック番号エリア３３の内容に変更し、このブロックの更新カウンタ３１を『１』インクリメントする。すなわち、未使用ブロックの最後に今削除したファイル４を接続するわけである。このようにして、更新カウンタ３１を進めながら何度もファイルの更新，削除を実行して行くうちに、更新カウンタ３１が１万回に接近する。

次に更新カウンタ３１が１万回に到達した場合のアクセス処理について説明する。

まず、ポインタブロック1aのスタートブロック番号OSBの内容が示しているブロックBLOCKのチェーンブロックエリア35の内容を新規のスタートブロック番号OSBとする。次いで、このブロック直前のディレクトリブロック30の更新カウンタ31の情報以外の内容を転送する。そして、ポインタブロック1aのディレクトリDBに新規のディレクトリブロック番号を書き込み、ポインタブロック1aの書き換え回数WCNTおよび更新カウンタ31を『1』インクリメントする。

一方、ポインタブロック1aの書き換え回数WCNTは1万回を越えた場合は、予備ポインタブロックSPB1～SPB50のうち一番近い予備ポインタブロックへ書き換え回数WCNTの情報以外のデータを転送し、新規のポインタブロックの書き換え回数WCNT（0000_{16}）を『1』インクリメントして『0001_{16}』に設定する。この場合、破棄されたポインタブロック1aの書き換え回数WCNTは1万回以上となり、新のポインタブロック1aの書き換え回数WCNTは1万回以下となる。このようにして、ディレクトリブロック30およびポインタブロック1aの書き込み削除を管理する。また削除されたファイルが使用していたブロックは未使用ブロックの一番最後に回される。これは、未使用ブロックの使用回数を平均化するためである。しかし、使用されているファイルが更新されずにずっとそのままであると、そのファイルが使用しているブロックは更新回数がそのまま変化しない。例えば、最初に作成されたファイルがそのままずっと登録されたまま残っていると、他のブロックは更新回数が5000回以上なのに、このファイルだけは2回というようなアンバランスが生じる。そこで、EEPROM1の使用状態を平均化するための補正処理を行う。

補正処理起動条件は下記（a），（b）の場合においてである。

（a）ポインタブロック1aの書き換え回数WCNTの値が256の整数倍になった時点。

（b）ファイルを構成するブロックの更新カウンタ31の平均値が一番低い値と、未使用ブロックの更新カウンタ31の平均値との差が256を越えた時点。

すなわち、新規のファイルを作成したり、削除したりした後、ポインタブロック1aの書き換え回数WCNTを見て、ちょうど256の整数倍、2バイトの16進数の下桁が『00_{16}』になった時点で、ディレクトリブロック30に登録されている順にファイルを検査して行く。そして、ファイルを構成するブロックの更新カウンタ31を加算し、さらに構成するブロック数で除して更新平均値を算出する。次いで、次のファイルに対しても同様の更新平均値を出して比較し、低いものをその比較対照として残し、次のファイルの更新平均値と比較して行き、一番低い更新回数で構成されるファイルを探し出す。そして、未使用のブロックの更新回数の平均を計算し、平均更新回数の一番低いファイルとの差を算出する。

次に第6図（a）～（c）を参照しながら補正処理動作について説明する。

第6図（a）～（c）はこの発明による補正動作を説明する模式図であり、これらの図において、41はファイルで、ブロックBLOCK5～7で構成され、平均更新回数が最も低いものである。42は未使用ブロック群で、ポインタブロック1aの未使用スタートブロック番号OSBで指示される。未使用ブロック群42は、ブロックBLOCK50，10，11，18，55，80，81が1つのチェイン構造となっている。

まず、ファイル41のスタートブロックを未使用ブロック群42のブロックBLOCK81の後段に接続させるため、同図（a）に示すようにブロックBLOCK81のチェーンブロックエリア35の内容が『FF_{16}』から同図（b）に示すように、チェーンブロックエリア35の内容が『05_{16}』、すなわち、ファイル41のブロックBLOCK5を指示させ、さらに、ファイル41のブロックBLOCK5～7（ファイル41のチェーンブロックエリア35の内容が『FF_{16}』になる

まで）を未使用ブロック群４２のブロックＢＬＯＣＫ８１の後段に接続させ，同図（ｃ）に示すように，ディレクトリブロック３０のファイル４１のスタートポインタをブロックＢＬＯＣＫ５０にするとともに，終了ポインタをブロックＢＬＯＣＫ１１に変更する．次いで，未使用ブロック群４２の未使用のスタートブロック番号ＯＳＢを『12_{16}』，未使用のエンドブロック番号ＯＥＢを『$0B_{16}$』にする．これにより，登録されたファイルを構成するブロックに対しても再度使用可能となり，ＥＥＰＲＯＭ１全体のブロックが平均的に使用，更新されることになる．第７図は第１図（ａ）に示したＥＥＰＲＯＭ１のデータ書き込み制御動作を説明するためのフローチャートである．なお，(1)～(18)は各ステップを示す．

まず，ディレクトリブロック３０の空エリアを探して，新規のファイル名を書き込む(1)．次いで，未使用のスタートブロック番号ＯＳＢをＣＰＵ１１のアキュムレータＡＣＣに記憶させる(2)．アキュムレータＡＣＣが指示するブロックの書き換え回数ＷＣＮＴを＋１更新する(3)．ここで，書き換え回数ＷＣＮＴが１００００を越えたかどうかを判断し(4)，ＹＥＳならばアキュムレータＡＣＣの指示するブロックの継続ブロックエリアＣＢをアキュムレータＡＣＣに記憶し(5)，ステップ(3)に戻り，ＮＯならばディレクトリブロック３０のスタートブロック番号エリア（ＳＢ）３３にアキュムレータＡＣＣの内容を書き込む(6)．次いで，アキュムレータＡＣＣが指示するブロックのデータエリアにデータを書き込む(7)．ここで，書き込みデータがアキュムレータＡＣＣが指示するブロックの容量が２３５バイトを越えるかどうかを判断し(8)，ＹＥＳならばアキュムレータＡＣＣが指示するブロックの継続ブロックエリアＣＢをアキュムレータＢＣＣに記憶させる(9)．次いで，アキュムレータＢＣＣが指示するブロックの書き換え回数ＷＣＮＴを＋１更新する(10)．次いで，書き換え回数ＷＣＮＴが１００００を越えたかどうかを判断し(11)，ＹＥＳならばアキュムレータＢＣＣの指示するブロックの継続ブロックエリアＣＢを記憶させ(12)，ステップ(10)に戻り，ＮＯならばアキュムレータＡＣＣが指示するブロックの継続ブロックエリアＣＢにアキュムレータＢＣＣの内容を書き込み(13)，ステップ(7)に戻る．

一方，ステップ(8)の判断でＮＯの場合は，アキュムレータＡＣＣが指示する継続ブロックエリアＣＢを未使用のスタートブロック番号ＯＳＢに書き込む(14)．次いで，ポインタブロック１ａの書き換え回数ＷＣＮＴを＋１更新する(15)．次いで，アキュムレータＡＣＣが指示するブロックの継続ブロックエリアＣＢへ「FF_{16}」を書き込む(16)．そして，ディレクトリブロック３０の新ファイル位置のエンドブロック番号エリア３４へアキュムレータＡＣＣの内容を書き込む(17)．次いで，ディレクトリブロック３０の書き換え回数ＷＣＮＴを更新する(18)．

第８図はこの発明による補正制御動作手順を説明するためのフローチャートである．なお，(1)～(7)は各ステップを示す．

ポインタブロック１ａの書き換え回数ＷＣＮＴが２５６の整数倍であるかどうかを判断し(1)，ＮＯならばリターン（ＲＥＴＵＲＮ）し，ＹＥＳならばディレクトリブロック３０に登録された各ファイルを構成するブロックの更新カウンタ３１の平均値を算出して，最も更新回数が少ないファイルを探し出す(2)．次いで，未使用のブロックの更新回数の平均値を算出する(3)．次いで，未使用ブロックの更新カウンタの平均値からファイルを構成するブロックの更新カウンタの平均値の最小値を減算し，さらに減算値から２５６を差し引いた値が正かどうかを判断し(4)，ＮＯならばリターンし，ＹＥＳならば未使用ブロックの最後尾に該当するファイルのヘッドを接続させる(5)．次いで，接続したファイルの内容を未使用ブロックへ転送させ(6)，ディレクトリブロック３０にある接続したファイルのスタートポインタ，エンドポインタを変更し(7)，リターンする．

〔発明の効果〕

以上説明したように、この発明は記憶領域を複数に分割し、各ブロック毎に書き込み回数を記憶し、この書き込み回数に応じて書き込み頻度の低いブロックを未使用ブロックの先頭から書き込むとともに、書き込み頻度の低いブロックを未使用ブロックの最後尾に接続させるようにしたので、ＥＥＰＲＯＭに書き込まれるデータの消失を防止するとともに、ＥＥＰＲＯＭへの書き込み回数を平均化させることができる。またＥＥＰＲＯＭ上の各ブロックの書き換え頻度を平均化でき、書き換え寿命を大幅に延長できる優れた利点を有する。

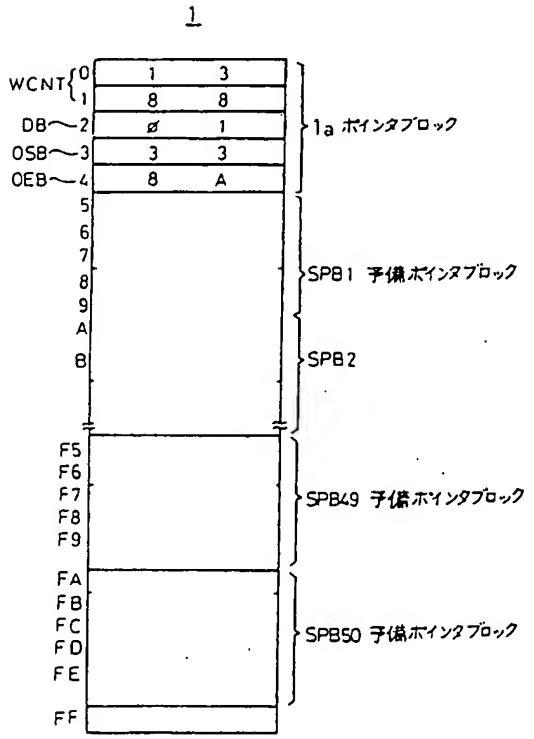
4．図面の簡単な説明

第１図（ａ）はこの発明の一実施例を示すプログラマブルリードオンリメモリへの書き込み回数管理方式を説明する模式図、第１図（ｂ）はこの発明の装置構成を説明するためのブロック図、第２図は第１図（ａ）に示すＥＥＰＲＯＭの構造を示す模式図、第３図は第２図に示す各ディレクトリブロック構造を説明する模式図、第４図は未使用のＥＥＰＲＯＭの状態を説明する模式図、第５図（ａ），（ｂ）はＥＥＰＲＯＭへの書き込み動作を説明する模式図、第６図（ａ）～（ｃ）はこの発明による補正処理動作を説明する模式図、第７図は第１図（ａ）に示したＥＥＰＲＯＭのデータ書き込み制御動作を説明するためのフローチャート、第８図はこの発明による補正制御動作手順を説明するためのフローチャートである。

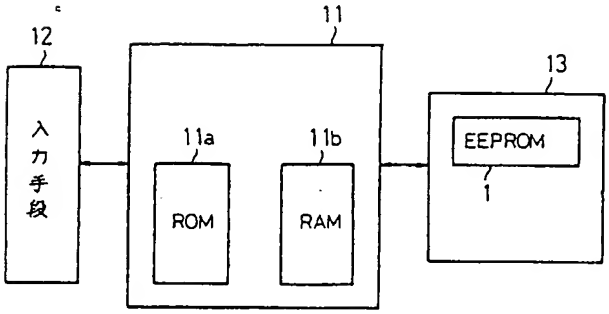
図中、１はＥＥＰＲＯＭ、１ａはポインタブロック、２１はブロック番号、３０はディレクトリブロック、３１は更新カウンタ、３２はファイル領域、３３はスタートブロック番号エリア、３４はエンドブロック番号エリア、３５はチェーンブロックエリア、４１はファイル、４２は未使用ブロック群である。

代理人　小　林　将　高

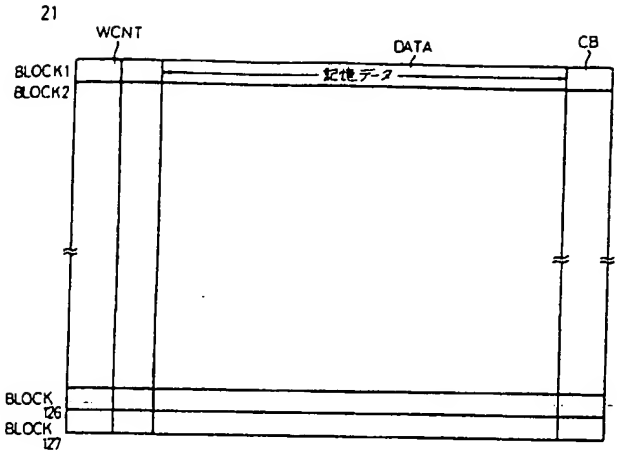
第　１　図　(a)



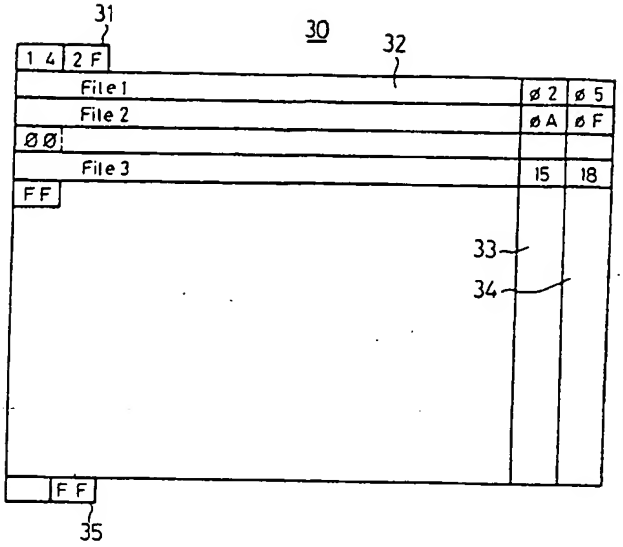
第　１　図　(b)



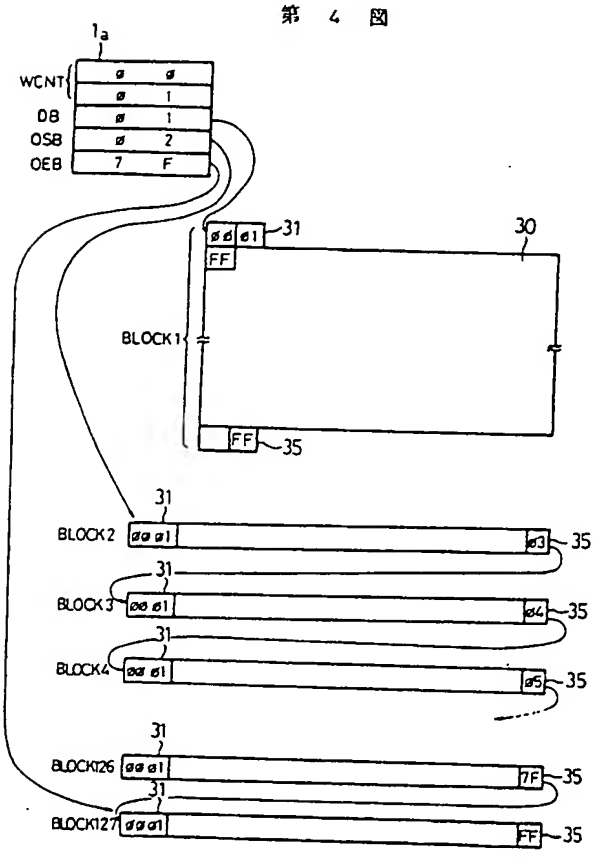
第 2 図



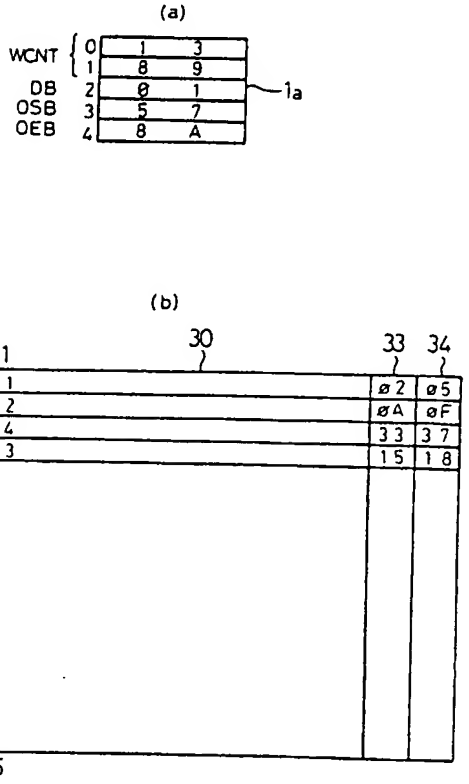
第 3 図



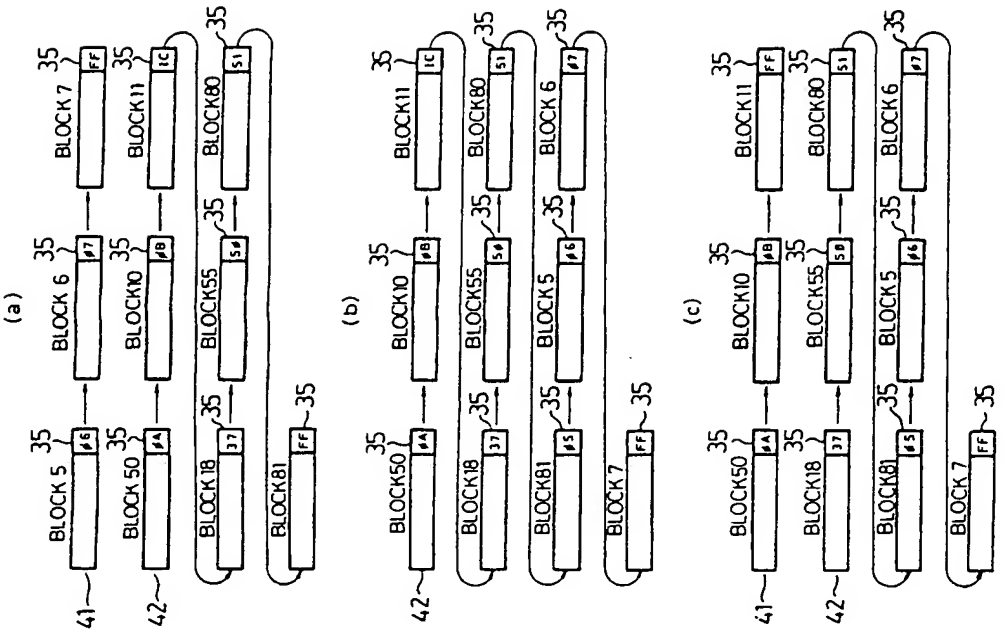
第 4 図



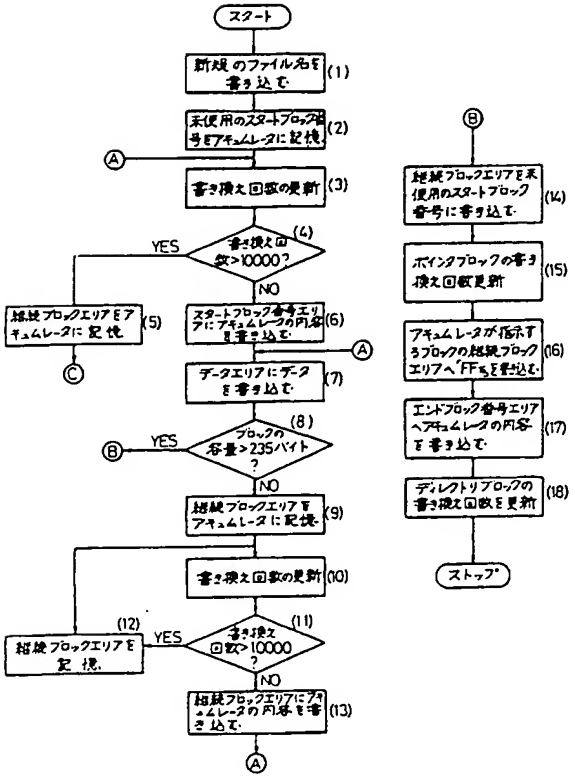
第 5 図



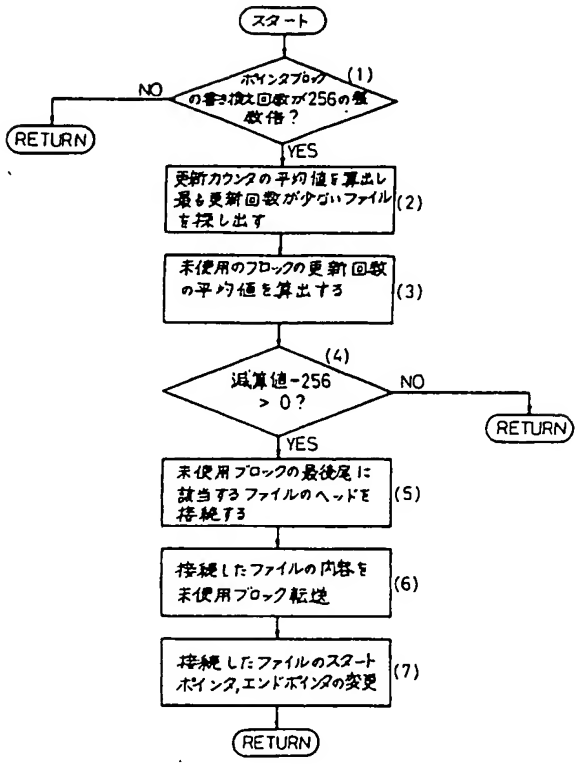
第 6 図


(a)
BLOCK 7
BLOCK11
BLOCK80
BLOCK 6
BLOCK10
BLOCK55
BLOCK 5
BLOCK 50
BLOCK18
BLOCK81
41
42
(b)
BLOCK11
BLOCK80
BLOCK 6
BLOCK10
BLOCK55
BLOCK 5
BLOCK50
BLOCK18
BLOCK81
BLOCK 7
42
(c)
BLOCK11
BLOCK80
BLOCK 6
BLOCK10
BLOCK55
BLOCK 5
BLOCK50
BLOCK18
BLOCK81
BLOCK 7
41
42

第 7 図


スタート
新規のファイル名を書き込む (1)
未使用のスタートブロック番号をアキュムレータに記憶 (2)
書き換え回数の更新 (3)
書き換え回数>10000? (4)
YES
NO
結線ブロックエリアをアキュムレータに記憶 (5)
スタートブロック番号エリアにアキュムレータの内容を書き込む (6)
データエリアにデータを書き込む (7)
ブロックの容量>235バイト? (8)
結線ブロックエリアをアキュムレータに記憶 (9)
書き換え回数の更新 (10)
書き換え回数>10000 (11)
結線ブロックエリアを記憶 (12)
結線ブロックエリアに書き込む (13)
結線ブロックエリアを未使用のスタートブロック番号に書き込む (14)
ポインタブロックの書き換え回数更新 (15)
アキュムレータが指示するブロックの結線ブロックエリアへFFを書き込む (16)
エンドブロック番号エリアへアキュムレータの内容を書き込む (17)
ディレクトリブロックの書き換え回数を更新 (18)
ストップ

第 8 図


スタート
ポインタブロックの書き換え回数が256の整数倍? (1)
NO
YES
RETURN
更新カウンタの平均値を算出し最も更新回数が少ないファイルを探し出す (2)
未使用のブロックの更新回数の平均値を算出する (3)
減算値-256>0? (4)
NO
YES
RETURN
未使用ブロックの最後尾に該当するファイルのヘッドを接続する (5)
接続したファイルの内容を未使用ブロック転送 (6)
接続したファイルのスタートポインタ,エンドポインタの変更 (7)
RETURN

Japanese Patent Application KOKAI No. 62-283496

Date of KOKAI: December 9, 1987

Title of the Invention: Write operation of a programable read only memory

Application No. 61-124731

filed May 31, 1986

Inventor: Shinichi Nakada

Applicants: Canon INC.

Specification

1. Title of the Invention:

Write operation of a programable read only memory

2. The Claim:

System of managing the write operations to the programable read only memory wherein data written in the memory area thereof can electrically be erasable; where

said memory area is divided into a plurality of blocks,

the number of times data was written into each block is stored,

data is written from blocks where the frequency of times data is written is low into unused blocks in order in the ascending order of the frequency of times data is written into the former blocks,

said blocks where the frequency of times data is written is low are connected behind said unused blocks.

3. Detailed Explanation of the Invention:

(Field of the invention)

This invention relates to the management of the number of time data can repetitively be written into an electrically erasable, programable read only memory.

(Background of the invention)

An EEPROM (electrically erasable and programable ROM) in the early time was small in capacity and operated with a great amount of peripheral circuits for a write operation. The peripheral circuits in the early time were operated in such an erase mode that all data stored in the EEPROM chip is erased at a time. Recently, EEPROM is increased in capacity and operated with fewer peripheral circuits even if it is connected to the address and data buses of the CPU. Any one byte of data can be erased from EEPROM right now. Due to the improvement described above, the random access memory (RAM) in the prior art can be replaced by the EEPROM for special purpose.

The memory card is used to keep the alphabetical characters, programs, and/or sentences stored after they are generated from a machine, typically a small personal computer or a word processor for Japanese characters. Since the memory card consists of a battery and an RAM, it can receive the alphabetical characters, programs, and/or sentences from the machine, when set in the machine, and also keep them stored after pulled out of the machine. If the memory card is built with an EEPROM, the battery may be omitted.

(Objective of the invention)

The number of times data can repetitively be written into the EEPROM is however limited unlike the RAM since there are no limitation on the RAM. This implies that data can disappear when written into the EEPROM beyond the limited number of times data can repetitively be written into the memory card consisting of an EEPROM. Some data is frequently replaced by new one, and other data is rarely replaced by new one. If the number of times the former is rewritten exceeds the limit on

the EEPROM, the contents of the EEPROM cannot be replaced any more even though the number of times the latter is rewritten into the EEPROM is within the limitation.

Since the present invention is disclosed to solve the above problem, the objective of the present invention is to provide the system of managing the number of times data can repetitively be written into the programable read only memory, whereby the life of the EEPROM can be extended through the protection of written data against disappearing, averaging of the number of times of write operations for every EEPROM cell, and averaging of the frequency of times of write operations for every EEPROM cell.

(Summary of the invention)

The present invention describes the system of managing the number of times data can repetitivery be written into the programable read only memory, which is characterized in that the memory area is divided into a plurality of blocks, the number of times data was written into each block, data of the respective blocks where the frequency of times data is written thereinto is low is written into unused blocks starting from the beginning thereof in an ascending order of said number of times data was written into each block, and said blocks where the frequency of times data is written thereinto is low are connected behind the unused blocks.

(Principle of operation)

In the present invention, the number of times data was written into each block of the memory area is stored, data of the blocks where the frequency of times data was written into each block is low is written into the unused blocks in an ascending order of the number of times data was written into

each block, and the blocks where the frequency of times data was written into each block is low are connected behind the unused blocks.

(Embodiments of the invention)

Figure 1(a) shows the pointer and its spare blocks which are used to manage the number of times data can be written into the programable read only memory in accordance with the present invention. In Figure 1(a), 1 indicates the EEPROM with a capacity of 32788 bytes by 8 bits, whereinto data can be written up to 10,000 times. EEPROM 1 consists of pointer block 1a and its spare blocks SPB1 through SPB50. Pointer block 1a contains 4 addresses, each consisting of one byte. Addresses "0" and "1" constitute a write counter (WCNT) consisting of 2 bytes and they can store the number of times data was repetitively written. They typically indicate "1388_{16}". Address "2" of pointer block 1a, consisting of one byte, stores directory DB, typically "01_{16}". Address "3" of pointer block 1a, consisting of one byte, stores unused start block address OSB, typically "33_{16}". Address "4" of pointer block 1a, consisting of one byte, stores unused end block address OEB, typically "08_{16}".

Figure 1(b) shows the block diagram of the system configuration as an embodiment in accordance with the present invention. In Figure 1(b), 11, 11a, and 11b indicate the CPU, ROM, and RAM, respectively. The system operation is controlled by the program specified by the flowcharts of Figures 7 and 8 which are being stored in ROM 11a. In addition, 12 indicates input means which specify the operations to write data into EEPROM 1 which has been installed in write data device 13 and to erase data from EEPROM 1. CPU 11 contains accumulators ACC

and BCC which perform the executions, respectively.

Figure 2 shows the configuration of a file in EEPROM 1 of Figure 1(a). In Figure 2, 21 indicates block addresses, typically BLOCK 1 through BLOCK 127. Each block typically consists of 256 bytes, where the leading 2 bytes store the number of times data was updated in the corresponding block. The succeeding 253 bytes keep data stored. The ending one byte thereof stores continue block area CB which indicates whether data being stored is complete or is continued to any other block. If data continues to any other block, continue block area CB stores the block address to which data continues. Unless data continues to any other block, continue block area CB stores FF_{16}.

Figure 3 shows the configuration of the directory block of Figure 2. In Figure 3, 30 indicates the directory block specified by said directory DB, 31 indicates the update counter to update the contents of said directory block 30, which typically consists of 2 bytes, 32 indicates the file area which stores the filename, typically consisting of 12 bytes, 33 indicates the start block address area (SB) which stores the start block address of the corresponding file, typically consisting of one byte, 34 indicates the end block address area (EB) which stores the end block address of the corresponding file, typically consisting of one bytes, and 35 indicates the chaining block area (CB) which stores information of whether any directory block follows directory block 30 or not, typically "FF_{16}". Directory block 30 typically consists of 15 file areas 32.

The structure of EEPROM 1 will be described hereafter referring to Figures 1(a) and 3.

If write counter WCNT in bytes 0 and 1 of pointer block 1a indicates typically "1388_{16}" as shown in Figure 1(a), they mean that data was updated 5,000 times. In Figure 1(a), directory DB stores "01_{16}". That is, the block address of directory block 30 specified by directory DB is "1", and update counter 31 of directory block 30 stores "$142F_{16}$". The contents of update counter 31 indicate that the contents of directory block 30 were updated 5167 times. In file 1 (FiLe) (filename) of file area 32, start block address area 33 is specified as "02_{16}" and end block address area 34 as "05_{16}". They mean that file 1 starts at block BLOCK 2 and terminates at block BLOCK 5. In file 2 of file area 32, start block address area 33 is specified as "$0A_{16}$" and end block address area 34 as "$0F_{16}$". They mean that file 2 starts at BLOCK 10 and terminates at block BLOCK 15. In file 3 (filename) of file area 32, start block address area 33 is specified as "15_{16}" and end block address area 34 as "18_{16}". They mean that file 3 starts at BLOCK 21 and terminates at BLOCK 24. Continue block area following file 3 of file area 33 is specified as "FF_{16}" and it means that file area 33 ends with file 3.

Figure 4 shows the configuration of the unused blocks in EEPROM 1, wherein the elements having appeared in Figures 1(a) and 3 are assigned by the same numbers.

For the unused blocks of EEPROM 1 in Figure 4, write counter WCNT of pointer block 1 is specified as "0001_{16}", directory DB thereof as "01_{16}", unused start block address OSB thereof as "02_{16}", and unused end block address OEB thereof as "$7A_{16}$". These bytes occupy addresses 0 through 4 of pointer block 1a. In block BLOCK 1 specified by directory DB, update counter 31 is specified as "0001_{16}", file 1 of file area 32 as

"FF_{16}", and chaining block area 35 as "FF_{16}". They indicate that EEPROM 1 is unused.

Start block address 0SB of pointer block 1a is specified as "02_{16}" and end block address 0EB thereof as "$7E_{16}$". Leading 2 bytes of blocks BLOCK 2 through BLOCK 127 are respectively specified as "0001_{16}". Chaining block areas 35 for blocks BLOCK 2 through BLOCK 126 are respectively specified as "03_{16}" through "$7E_{16}$", which indicate the continuation of blocks, and chaining block area 35 for block BLOCK 127 as "FF_{16}". Note that the chaining block area is the one byte located at the end of each block. As described above, blocks BLOCK 2 through BLOCK 127 are chained.

How to write data into EEPROM 1 will be described hereafter referring to Figures 3, 5(a), and 5(b).

Figures 5(a) and 5(b) show the pointer and directory blocks used for writing data into EEPROM 1, respectively. In Figures 5(a) and 5(b), the elements having appeared in Figures 1(a) and 3 are assigned by the same number. Assume that the memory contents are as shown in Figure 3 right before data is written into EEPROM 1.

Control finds "00_{16}" from leading bytes of file area 32 in a certain block BLOCK#. In Figure 3, "00_{16}" are located between files 2 and 3. At that time, control writes filename "file 4" there in 12 bytes. Referring to start block address 0SB of an unused block in pointer block 1a, control increments the contents of update counter 31 by one, or increments data in leading 2 bytes of block BLOCK 57_{16} specified by start block address 0SB. If the contents of update counter 31 exceed typically 10,000, control performs the same operations as above for the block BLOCK# specified by chaining block area 35 of

file 4 until control finds a block BLOCK# whose update counter 31 indicates a number of less than 10,000. At that time, control writes the block address of that block into start block address area OSB of pointer block 1a and also writes data of file 4 into block BLOCK 87 (253 bytes). If block BLOCK 87 overflows, control increments by one the contents of update counter 31 for block BLOCK# specified by chaining block area 35 of block BLOCK 87. Thereafter, control checks if the contents of that update counter 31 exceed typically 10,000. If update counter 31 indicates a number of greater than 10,000, control finds a block BLOCK# whose update counter 31 indicates a number of less than 10,000, and control writes the block address of that block BLOCK# into chaining block area 36 of block BLOCK# whereinto data was written right before. Data is written into blocks in this manner so that blocks BLOCK# whereinto data was repetitively written 10,000 times or more can be rejected. The same operation is repeated until no data is written into EEPROM 1. When data is written into a last block, control writes new unused start block address OSB in place of the contents of chaining block area 35 of said last block BLOCK#. Thereafter control increments by one the contents of write counter WCNT of pointer block 1a so that said write counter WCNT indicates "1389_{16}", and enters "FF_{16}" into chaining block area 35 of said last block BLOCK#. Tnereafter, control writes the address of said last block BLOCK# into end block address area 34 which stores the last block address of directory block 30, and increments by one the contents of update counter 31. The contents of update counter 31 then becomes "1430_{16}" as shown in Figure 5(b). At that time, start block address area 33 of file 4 is specified as "33_{16}" and end block address area 34 thereof

as "37_{16}".

How to delete file 1 from EEPROM 1 will be described hereafter referring to Figures 5(a) and 5(b).

Control finds file 1 from block Block 1 since block BLOCK 1 is used as directory block 30, and then control specifies leading 2 bytes of file area 32 as "00_{16}". Control then increments by one the contents of update counter 31 of directory block 30. Referring to the contents of both start block address area 33 and end block address area 34 of file 1, control changes the contents (which were "FF_{16}" right before the deletion of file 1) of chaining block area 35 of the block specified by end block address OEB of pointer block 1a into the contents of start block address area 33. Control then increments by one the contents of update counter 31 of that block. This operation implies that control connects file 4 (which has been deleted) behind unused blocks. In this manner, files are repetitively updated and deleted while the contents of update counter 31 are incremented, and finally the contents of update counter 31 approach 10,000.

How to access EEPROM 1 when the contents of update counter 31 approach 10,000 will be described hereafter.

Control first sets new start block address OSB at chaining block area 35 of the block BLOCK# specified by start block address OSB of pointer block 1a. Control then transfers any data other than the contents of update counter 31 of directory block 30 located right before said block specified by start block address OSB of pointer block 1a. Thereafter, control writes new directory block address into directory DB of pointer block 1a, increments by one the contents of write counter WCNT of pointer block 1a, and also increments by one the contents of

update counter 31 thereof.

If the contents of write counter WCNT of pointer block 1a exceed 10,000, control transfers any other data than the contents of write counter WCNT to one of spare pointer blocks SPB1 through SPB50 in the nearest location, and increments by one the contents ("0000_{16}") of write counter WCNT of new pointer block so that the contents of write counter WCNT become "0001_{16}". At that time, the contents of write counter WCNT of pointer block 1a, which have been disregarded, become greater than 10,000, and the contents of write counter WCNT of new pointer block 1a become less than 10,000. The delete and write operations for both directory block 30 and pointer block 1a are managed in this manner. Control moves the block used by deleted life behind the last unused block so that the frequency of times for the use of unused blocks can be averaged. Unless a used file is updated, the number of times data was updated in the block used by that file does not change. If any file remains cataloged without its use after created, the number of times data was updated in that file is in some cases much less than the number of times data was updated in the other block. For instance, the former is 2 while the latter is 5,000 or more. The number of times data was written into each block of EEPROM 1 is thus to be averaged.

The complementary processing for the averaging can start under condition (a) or (b) described below.

(a). The contents of write counter WCNT of pointer block 1a become a multiple of 256.

(b). The difference between the lowest average value for the contents of update counter 31 of the blocks, each of which constitutes a file, and the average value for the contents of

update counter 31 of the unused blocks exceeds 256.

After creating a new file or deleting a file, control checks if the contents of write counter WCNT of pointer block 1a become a multiple of 256 or if the low order byte of loading 2 bytes becomes "00_{16}". At the time the above condition becomes valid, control performs the test for the files in accordance with the order that the files have been cataloged into directory block 30. Control accumulates the contents of update counters 31 for the blocks which constitute a file, and divides the sum of these contents by the number of blocks which constitute that file so that the average number of times data was updated could be obtained. Thereafter, control calculates the average number of times data was updated in each of the other files. Control then compares the averages among the others so that the lowest number of times data was updated could be used as a reference for the comparison. Control then finds the file whose number of times data was updated is the lowest. Control calculates the average number of times data was updated in unused blocks, and also calculates the difference between this average and the reference obtained before.

The complementary processing will be described referring to Figures 6(a) through 6(c).

Figures 6(a) through 6(c) show the configuration of blocks when the complementary processing is carried out for the averaging. In Figures 6(a) through 6(c), 41 indicates the file consisting of blocks BLOCK 5 through BLOCK 7, and the average number of times data was updated therein is lowest. In addition, 42 indicates a group of unused blocks, which is specified by unused start block address OSB of pointer block

1a. Unused block group 42 consists of chaining blocks BLOCK 50, BLOCK 10, BLOCK 11, BLOCK 18, BLOCK 55, BLOCK 80, and BLOCK 81.

Control connects the start block of file 41 behind block BLOCK 81 of unused block group 42 in the following manner. That is, the contents of chaining block area 35 of block BLOCK 81 are changed from "FF_{16}", (shown in Figure 6(a)) to "05_{16}" (shown in Figure 6(b)) so that chaining block area 35 indicates block BLOCK 5 of file 41. Until the contents of chaining block area 35 of file 41 become "FF_{16}", control connects blocks BLOCK 5 through BLOCK 7 of file 41 behind block BLOCK 81 of unused block group 42. At that time, control specifies the start pointer for file 41 in directory block 30 as block BLOCK 50 and the end pointer therefor as block BLOCK 11, as shown in Figure 6(c). Thereafter, control specifies unused start block address OSB of unused block group 42 as "12_{16}" and unused end block address OEB thereof as "$0B_{16}$". All blocks used as part of a cataloged file can thus be used again, and blocks in EEPROM 1 can equally be used and updated.

Figure 7 shows the flowchart of the write operation for EEPROM of Figure 1(a). Items 1 through 18 indicate the processing steps.

Control finds an unused area from directory block 30, and writes a new filename there. (See step 1.) Control stores unused start block address OSB to accumulator ACC for CFULL. (See step 2.) Control increments by one the contents of write counter WCNT specified by accumulator ACC. (See step 3.) Control checks if the contents of write counter WCNT exceed 10,000. (See step 4.) If the response is YES, control stores continue block area CB of the block specified by accumulator

ACC in accumulator ACC and then returns to step 3. (See step 5.) If the response is No, control writes the contents of accumulator ACC into start block address area SB of directory block 30. (See step 6.) Thereafter, control writes data into the data area of the block specified by accumulator ACC. (See step 7.) Control then checks if data being written into the block specified by accumulator ACC exceeds the capacity of the block specified by accumulator ACC or 235 bytes. (See step 8.) If the response is YES, control stores continue block area CB of the block specified by accumulator ACC into accumulator BCC. (See step 9.) Next, control increments by one the contents of write counter WCNT of the block specified by accumulator BCC. (See step 10.) Thereafter, control checks if the contents of write counter WCNT exceeds 10,000. (See step 11.) If the response is YES, control stores continue block area CB of the block specified by accumulator BCC, and then returns to step 10. If the response is NO, control writes data of accumulator BCC into continue block area CB of the block specified by accumulator ACC, and returns to step 7.

If the response is NO at step 8, control writes continue block area CB specified by accumulator ACC into unused start block address OSB, (See step 14.) Next, control increments by one the contents of write counter WCNT of pointer block 1a. (See step 15.) Thereafter, control writes "FF_{16}" into continue block area CB specified by accumulator ACC. (See step 16.) Then, control writes data of accumulator ACC into end block address area 34 on a location allocated for a new file in directory block 30. (See step 17.) Control updates the contents of write counter WCNT of directory block 30. (See step 18.)

Figure 8 shows the flowchart of the complementary processing performed in accordance with the present invention, where items 1 through 7 indicate steps 1 through 7, respectively.

Control checks if the contents of write counter WCNT of pointer block 1a are a multiple of 256. (See step 1.) If the response is NO, control returns to the start. If the response is YES, control calculates the average for the contents of update counter 31 of the blocks constituting each file cataloged in directory block 30, and finds the file where the number of times data was updated is the lowest. (See step 2.) Control then calculates the average for the number of times data was updated in the unused blocks. (See step 3.) Thereafter, control subtracts the lowest value of the averages for the contents of update counters in the blocks which constitute the corresponding file from the average for the contents of the update counters of the unused blocks. Thereafter, control checks if the result of the subtraction is greater than 256. (See step 4.) If the response is NO, control returns to the start. If the response is YES, control connects the head of the corresponding file behind the unused blocks. (See step 5.) Thereafter, control transfers the contents of the file connected behind the unused blocks to one of the unused blocks. (See step 6.) Control alters the start and end pointers of the connected file in directory block 30. (See step 7.) Control then returns to the start.

(Advantages of the invention over the technology in the prior art)

In accordance with the present invention, as described above, the memory area is divided into a plurality of blocks,

the number of times data was written is stored for each block, data of blocks where the frequency of times data is written is low is written into the unused blocks starting with the beginning of the unused blocks in accordance with the ascending order of the number of times data was written, and the block where the frequency of times data can be rewritten is the lowest is connected behind the unused blocks. The above operations protects EEPROM against disappearing of data, and averages the number of times data was written into EEPROM. This invention averages the frequency of times data is written into each block of EEPROM, and the life determined by the number of times data can be written can be extended.

4. Brief description of the drawings

Figure 1(a) shows the pointer and its spare blocks used to manage the number of times data can be written into EEPROM in accordance with the present invention.

Figure 1(b) shows the block diagram of the system configuration as an embodiment in accordance with the present invention.

Figure 2 shows the configuration of a file in EEPROM.

Figure 3 shows the configuration of the directory block of Figure 2.

Figure 4 shows the configuration of the unused blocks in EEPROM.

Figures 5(a) and 5(b) show the pointer and directory blocks used for writing data into EEPROM, respectively.

Figures 6(a) through 6(c) show the configiration of blocks when the complementary operation is carried out for obtaining the average number of times data was written into blocks.

Figure 7 shows the flowchart of the write operation for

EEPROM 1 of Figure 1(a).

Figure 8 shows the flowchart of the complementary processing performed in accordance with the present invention.

1....EEPROM

1a...pointer block

21...block address

30...directory block

31...update counter

32...file area

33...start block address area

34...end block address area

35...chaining block area

41...file

42...unused block group

Figure 1(a)

1a: pointer block

SPB1: Spare Pointer Block

SPB2:

SPB49: Spare Pointer Block

SPB50: Spare Pointer Block

Figure 1(b)

12: Input Means

Figure 2

BLOCK 1 Stored Data

Figure 7

START

(1) WRITE NEW FILENAME.

(2) STORE UNUSED START BLOCK ADDRESS IN ACCUMULATOR.

(3) UPDATE NUMBER OF TIMES DATA WAS REPETITIVELY WRITTEN.

(4) NUMBER OF TIMES DATA WAS WRITTEN > 10,000?

(5) STORE CONTINUE BLOCK AREA IN ACCUMULATOR ACC.

(6) WRITE DATA OF ACCUMULATOR ACC INTO START BLOCK ADDRESS AREA.

(7) WRITE DATA INTO DATA AREA.

(8) CAPACITY OF BLOCK > 235 BYTES?

(9) STORE BLOCK AREA IN ACCUMULATOR.

(10) UPDATE NUMBER OF TIMES DATA WAS REPETITIVELY WRITTEN.

(11) NUMBER OF TIMES DATA WAS WRITTEN > 10,000?

(12) STORE CONTINUE BLOCK AREA IN ACCUMULATOR BCC.

(13) WRITE DATA OF ACCUMULATOR BCC INTO CONTINUE BLOCK AREA.

(14) WRITE DATA OF CONTINUE BLOCK AREA INTO UNUSED START BLOCK ADDRESS.

(15) UPDATE NUMBER OF TIMES POINTER BLOCK WAS REPETITIVELY WRITTEN

(16) WRITE "FF_{16}" INTO CONTINUE BLOCK AREA OF BLOCK SPECIFIED BY ACCUMULATOR ACC.

(17) WRITE DATA OF ACCUMULATOR ACC INTO END BLOCK ADDRESS AREA.

(18) UPDATE NUMBER OF TIMES DIRECTORY BLOCK WAS REPETITIVELY WRITTEN.

STOP

Figure 8

START

(1) NUMBER OF TIMES POINTER BLOCK WAS REPETITIVELY WRITTEN TO BE A MULTIPLE OF 256?

(2) FIND FILE WHERE NUMBER OF TIMES DATA WAS WRITTEN IS

LOWEST AFTER CALCULATING AVERAGE FOR CONTENTS OF UPDATE COUNTER.

(3) CALCULATE AVERAGE FOR NUMBER OF TIMES DATA WAS UPDATED IN UNUSED BLOCKS.

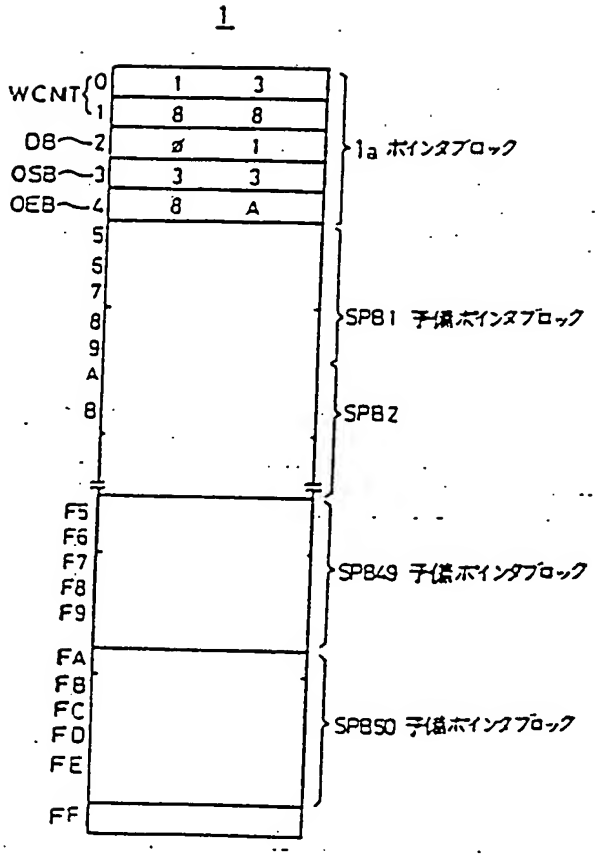
(4) SUBTRACTED VALUE -256 > 0?

(5) CONNECT HEAD OF CORRESPONDING FILE BEHIND UNUSED BLOCKS.

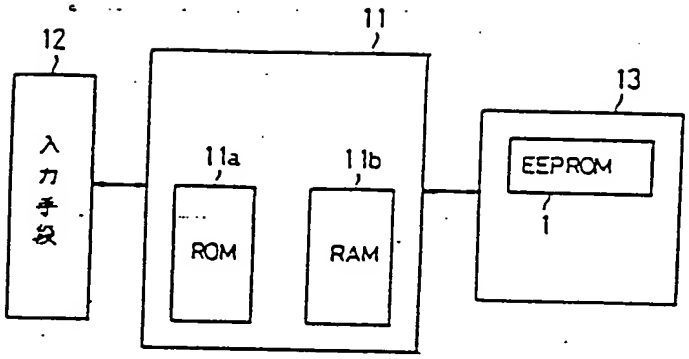
(6) TRANSFER CONTENTS OF CONNECTED FILE TO UNUSED BLOCK.

(7) ALTER START AND END POINTERS OF CONNECTED FILE.

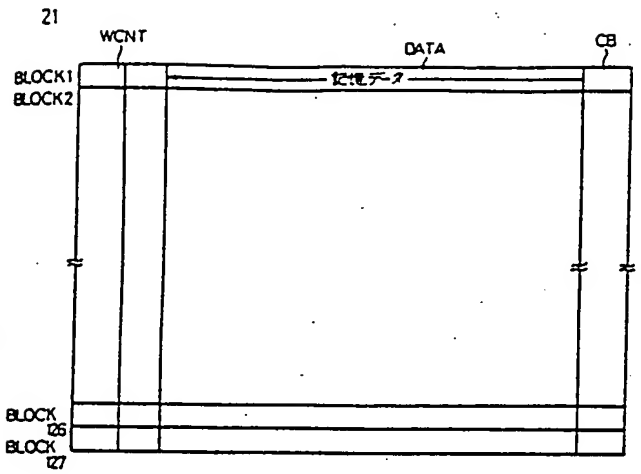
第 1 図 (a)


1
WCNT
0
1
DB
2
OSB
3
OEB
4
1 3
8 8
ø 1
3 3
8 A
1a ポインタブロック
5
6
7
8
9
A
B
SP81 子供ポインタブロック
SP82
F5
F6
F7
F8
F9
SP849 子供ポインタブロック
FA
FB
FC
FD
FE
SP850 子供ポインタブロック
FF

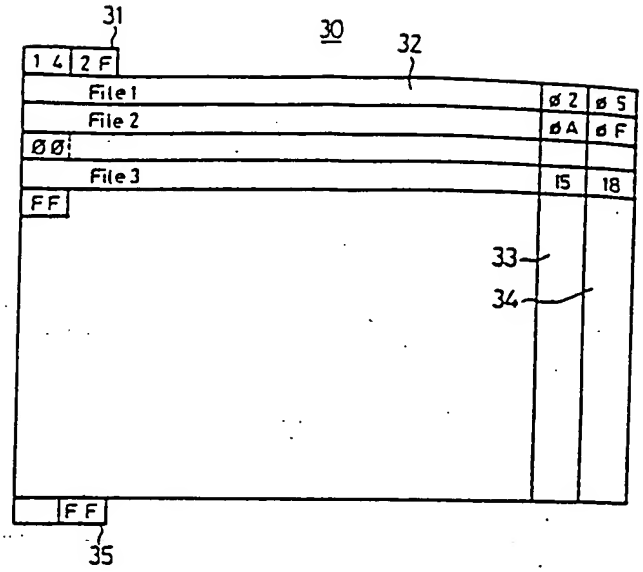
第 1 図 (b)


12
入力手段
11
11a
ROM
11b
RAM
13
EEPROM
1

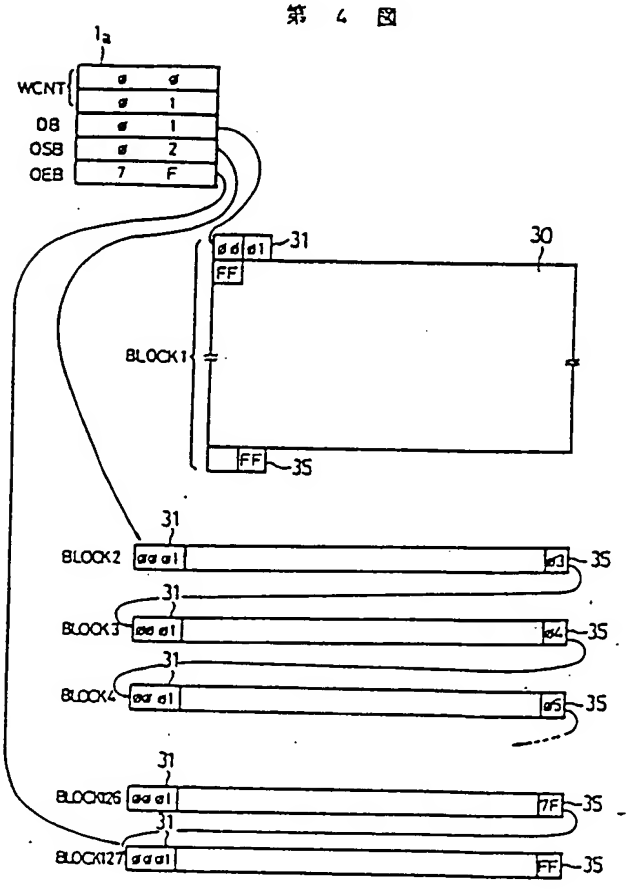
第 2 図



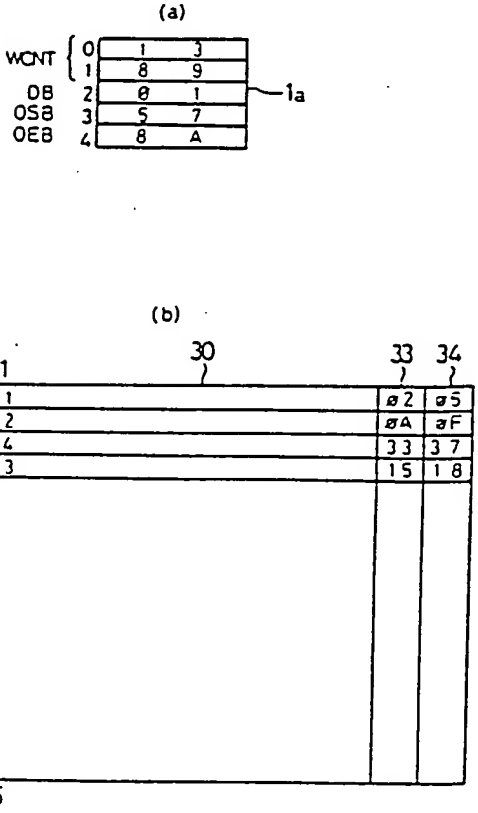
第 3 図



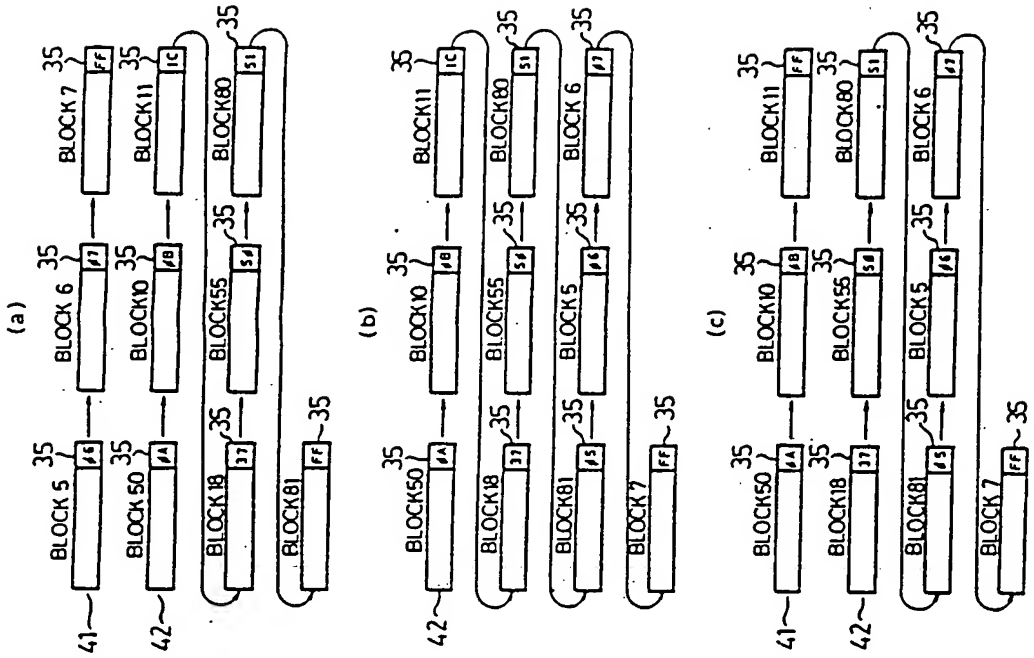
第 4 図



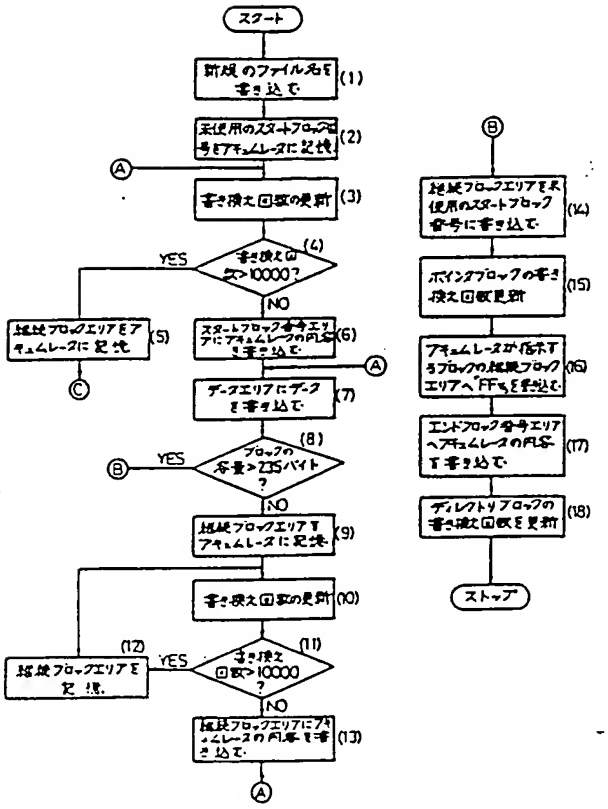
第 5 図



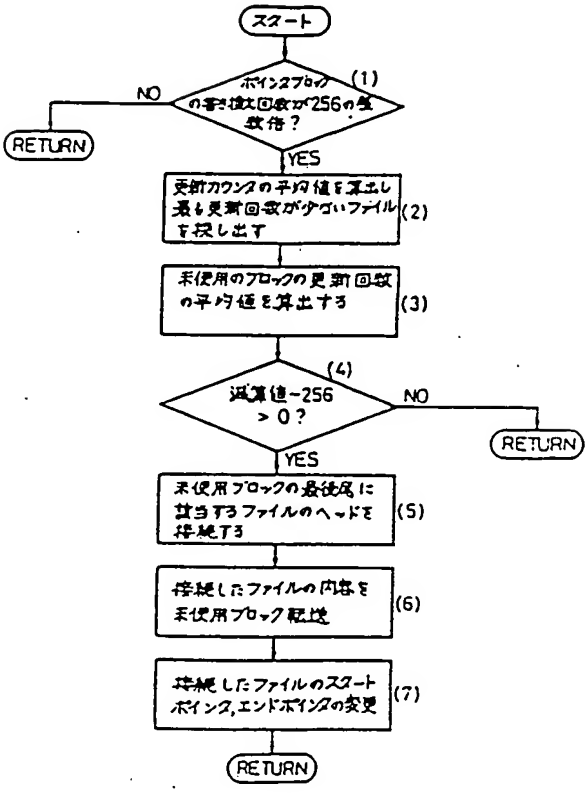
第 6 図


(a)
BLOCK 7
BLOCK 6
BLOCK 5
BLOCK11
BLOCK10
BLOCK50
BLOCK80
BLOCK55
BLOCK18
BLOCK81
(b)
BLOCK11
BLOCK10
BLOCK50
BLOCK80
BLOCK55
BLOCK18
BLOCK 6
BLOCK 5
BLOCK81
BLOCK 7
(c)
BLOCK11
BLOCK10
BLOCK50
BLOCK80
BLOCK55
BLOCK18
BLOCK 6
BLOCK 5
BLOCK81
BLOCK 7
35
41
42

第 7 図


スタート
新規のファイル名を書き込む (1)
未使用のスタートブロック番号をアキュムレータに記憶 (2)
書き換え回数の更新 (3)
書き換え回数>10000? (4)
YES
NO
連続ブロックエリアをアキュムレータに記憶 (5)
スタートブロック番号エリアにアキュムレータの内容を書き込む (6)
データエリアにデータを書き込む (7)
ブロックの容量>235バイト? (8)
連続ブロックエリアをアキュムレータに記憶 (9)
書き換え回数の更新 (10)
書き換え回数>10000? (11)
連続ブロックエリアを記憶 (12)
連続ブロックエリアにアキュムレータの内容を書き込む (13)
連続ブロックエリアを未使用のスタートブロック番号に書き込む (14)
ポインタブロックの書き換え回数更新 (15)
アキュムレータが指示するブロックの連続ブロックエリアへFFを書き込む (16)
エンドブロック番号エリアへアキュムレータの内容を書き込む (17)
ディレクトリブロックの書き換え回数を更新 (18)
ストップ
A
B
C

第 8 図


スタート
ポインタブロックの書き換え回数が256の整数倍? (1)
NO
YES
RETURN
更新カウンタの平均値を算出し最も更新回数が少ないファイルを探し出す (2)
未使用のブロックの更新回数の平均値を算出する (3)
減算値-256>0? (4)
NO
YES
RETURN
未使用ブロックの最後尾に該当するファイルのヘッドを接続する (5)
接続したファイルの内容を未使用ブロックに転送 (6)
接続したファイルのスタートポインタ,エンドポインタの変更 (7)
RETURN